Appendix

```
/*****************************************************************************
 *
 *
 *
 *
 * Description: A Palm database application used tracking the menstrual cycle of
 *              female Palm User.
 *
 *
 *
 *****************************************************************************/
//       1         2         3         4         5         6         7         8
//345678901234567890123456789012345678901234567890123456789012345678901234567890

#define NON_INTERNATIONAL

#include <Pilot.h>
#include <SysEvtMgr.h>

#include "PPatrol.h"
#include "Calendar.h"
#include "MyUtilities.h"

#include "pPatrolRsc.h"        // resource definitions (created by Constructor)


/*****************************************************************************
 * Internal structures definitions
 *****************************************************************************/
typedef struct
{
       DateType         theDate;                          // Date data was written
       int              theFlow;
       int              theMood;
       Boolean          theLast;
       Boolean          theFirst;
       char             theNotes[0];
} PackedData,       *PkdDataPtr;

#define DataSize    sizeof(PackedData)         // Min PackedData structure size


typedef struct
{
       Boolean          nextPeriod;
       Boolean          lastMissing;
       DateType         installedDate;          // Date application was installed
} AppPreferences;


/*****************************************************************************
 *  Global variables for this application
 *****************************************************************************/
AppPreferences       Prefs;                          // Preferences information

DateType             Nuday;                       // Date when device powered on
DateType             Today;
DateFormatType       DisplayDate;               // Format to use displaying date
DateFormatType       DisplayLongDate;           // Format to use displaying date

DateType             FirstDate;
DateType             FirstDays[TWELVE];
short                MonthDays[TWELVE];
short                GoodMonths;
short                SelectedMonths;        // Number of months selected by User
```

```
DmOpenRef           pPatrolDB;              // Handle for application's database

// These variables contain particular transaction values after unpacking a check
DateType            TheDate;                            // Date data was written
int                 TheFlow;
int                 TheMood;
Boolean             TheLast;
Boolean             TheFirst;
CharPtr             TheNotes;
CharPtr             TheOther;

Word                DailyFlow;            // Number indicating daily flow choice
Word                DailyMood;            // Number indicating daily mood choice
Word                RecordNumber;             // Record number used for whatever
Word                CurrentRecord;                // Index of the current record
Word                TopVisibleRecord;            // Top record in register table
Word                StartingDayOfWeek;         // Day of week the week starts on

short               AverageDays;
long                NumberDays;

int                 DailyFlowBMP;
int                 DailyMoodBMP;


#pragma mark -----Utilities----
/***************************************************************************
 *
 * Function:    CreateApplicationDatabase
 *
 * Description: This routine opens the application's database.  If the database
 *              does not exist, it will first create it and then open it.
 *
 * Parameters:  None
 *
 * Returns:     Nothing
 *
 * History:     09/10/00 - Initial creation of function for pPatrol project.
 *
 ***************************************************************************/
static void CreateApplicationDatabase(void)
{
       Word              error;                                    // Error code


       pPatrolDB = DmOpenDatabaseByTypeCreator(AppDbType, AppCreator, dmModeReadWrite);

       if( !pPatrolDB )                   // Database doesn't exist, so create it now
       {
               error = DmCreateDatabase(0, "pPatrolDB", AppCreator, AppDbType, false);
               ErrFatalDisplayIf(error, "Can't create new database."); // Check fatal error

               // Try opening the application database again
               pPatrolDB = DmOpenDatabaseByTypeCreator(AppDbType, AppCreator, dmModeReadWrite);

               MarkDatabaseAsDirty(pPatrolDB);          // Set dirty flag for this database

               CreateAppInfoBlock(pPatrolDB);          // Create and Initialize AppInfoBlock
       }
} // end of CreateApplicationDatabase


#pragma mark ------------------
/***************************************************************************
 *
 * Function:    MakeNewRecord
 *
 * Description: Create a new first record checkbook register entry on each call.
 *
 * Parameters:  None
 *
 * Returns:     True if a record was successfully created.
 *
 * History:     09/10/00 - Initial creation of function for pPatrol project.
 *
 ******************************************************************************/
static void MakeNewRecord(void)
{
        PackedData          patrol;
        VoidHand            nuData;                         // Handle for current record
        VoidPtr             pVoid;
        UInt                index;
        Err                 error;


        index = 0;                              // Add new record at beginning of database
        nuData = DmNewRecord(pPatrolDB, &index, DataSize);      // Get handle to record

        if( nuData )
        {
                patrol.theDate = Nuday;
                patrol.theFlow = NothingSelected;
                patrol.theMood = NothingSelected;
                patrol.theLast = false;
                patrol.theFirst = false;
                StrCopy(patrol.theNotes, "" + nullChr);

                pVoid = MemHandleLock(nuData);          // Lock handle and get pointer to data

                error = DmWrite(pVoid, 0, &patrol, DataSize);           // Write data record
                ErrFatalDisplayIf(error, "Can't write to new record");  // Check fatal error

                MemHandleUnlock(nuData);                // Finished, so unlock memory chunk

                CurrentRecord = index;                  // Remember index of current record
        }

        // Release record to database manager.  The 'true' value indicates this record
        // contains 'dirty' data. DmReleaseRecord will set the record's dirty flag and
        // update the database modification count.
        DmReleaseRecord(pPatrolDB, index, true);
} // end of MakeNewRecord


/******************************************************************************
 *
 * Function:    SaveCurrentRecord
 *
 * Description: Retrieve the current record and update the record's information.
 *
 * Parameters:  recordNum -> Number of the record to store.
 *
 * Returns:     Nothing
 *
 * History:     09/10/00 - Initial creation of function for pPatrol project.
 *
 ******************************************************************************/
static void SaveCurrentRecord(Word recordNum)
{
        VoidHand            nuData;                         // Handle for current record
        VoidPtr             pVoid;
        UInt                attributes;
        UInt                length, offset;


        // Sanity check, is passed record number within the number of database records
        if( recordNum < 0 || recordNum > DmNumRecords(pPatrolDB) ) return;

        // Calculate actual size of updated record  -  the 2 is for string terminators
        length = sizeof(TheDate) +  sizeof(TheFlow) + sizeof(TheMood) +
                                        sizeof(TheLast) + sizeof(TheFirst) +
                                        StrLen(TheNotes) + StrLen(TheOther) + 2;
        nuData = DmGetRecord(pPatrolDB, recordNum);      // Get handle for this record

        if( MemHandleResize(nuData, length) == 0 )
        {
                offset = 0;                         // Make sure offset always begins at zero

                pVoid = MemHandleLock(nuData);      // Lock handle and get pointer to record

                DmWrite(pVoid, offset, &TheDate, sizeof(TheDate));
                offset += sizeof(TheDate);

                DmWrite(pVoid, offset, &TheFlow, sizeof(TheFlow));
                offset += sizeof(TheFlow);

                DmWrite(pVoid, offset, &TheMood, sizeof(TheMood));
                offset += sizeof(TheMood);

                DmWrite(pVoid, offset, &TheLast, sizeof(TheLast));
                offset += sizeof(TheLast);

                DmWrite(pVoid, offset, &TheFirst, sizeof(TheFirst));
                offset += sizeof(TheFirst);

                DmStrCopy(pVoid, offset, TheNotes);
                offset += StrLen(TheNotes) + 1;

                DmStrCopy(pVoid, offset, TheOther);

                MemHandleUnlock(nuData);                // Finished, so unlock memory chunk

                MarkDatabaseAsDirty(pPatrolDB);         // Set dirty flag for this database

                DmRecordInfo(pPatrolDB, recordNum, &attributes, NULL, NULL);   // Attributes

                attributes |= dmRecAttrDirty;             // Set dirty flag for this record

                DmSetRecordInfo(pPatrolDB, recordNum, &attributes, NULL);
        }

        DmReleaseRecord(pPatrolDB, recordNum, true);      // See note in MakeNewRecord
} // end of SaveCurrentRecord


/**************************************************************************
 *
 * Function:    FetchCurrentRecord
 *
 * Description: Retrieves a data record from the database, unpacks it and places
 *              the data in a usable data structure.
 *
 * Parameters:  recordNum -> Number of the record to retrieve.
 *
 * Returns:     Nothing
 *
 * History:     09/10/00 - Initial creation of function for pPatrol project.
 *
 **************************************************************************/
static void FetchCurrentRecord(Word recordNum)
{
        PackedData        *patrol;
        VoidHand          moniker;


        // Sanity check, is passed record number within the number of database records
        if( recordNum < 0 || recordNum > DmNumRecords(pPatrolDB) ) return;

        moniker = DmQueryRecord(pPatrolDB, recordNum);       // Get handle for record

        if( moniker )
        {
                patrol = MemHandleLock(moniker);      // Lock handle and get pointer to data

                TheDate = patrol->theDate;
                TheFlow = patrol->theFlow;
                TheMood = patrol->theMood;
                TheLast = patrol->theLast;
                TheFirst = patrol->theFirst;

                TheNotes = patrol->theNotes;
                TheOther = TheNotes + StrLen(TheNotes) + 1;

                MemHandleUnlock(moniker);                // Finished, so unlock memory chunk
        }
} // end of FetchCurrentRecord


/***************************************************************************
 *
 * Function:    GetNumberOfRecords
 *
 * Description: This routine gets the total number of records in the currently
 *              active account/database.
 *
 * Parameters:  None
 *
 * Returns:     Nothing
 *
 * History:     09/10/00 - Initial creation of function for pPatrol project.
 *
 ***************************************************************************/
static void GetNumberOfRecords(void)
{
        CharPtr         pText;
        Word            numRecords;


        pText = MemPtrNew(50);
        numRecords = DmNumRecords(pPatrolDB);                   // Get number of records

        StrCopy(pText, "There are ");
        StrIToA(pText + StrLen(pText), numRecords);
        StrCat(pText, " records in the pPatrol DB.");

        FrmCustomAlert(InformationAlert, pText, NULL, NULL);
        MemPtrFree(pText);
} // end of GetNumberOfRecords


/***************************************************************************
 *
 * Function:    CompareSavedRecords
 *
 * Description: This routine compares the current date with those already saved.
 *
 * Parameters:  None
 *
 * Returns:     True if a match is found otherwise false.
 *
 * History:     09/20/00 - Added code to support database for pPatrol project.
 *
 ***************************************************************************/
static Boolean CompareSavedRecords(void)
{
        Boolean         matched = false;
        Word            numRecords, recordNum;


        numRecords = DmNumRecords(pPatrolDB);        // Get number of database records

        for( recordNum = numRecords - 1; (short)recordNum >= 0; recordNum-- )
        {
                FetchCurrentRecord(recordNum);

                if( CompareTwoDates(Nuday, TheDate) == 0 )            // Are they the same?
                {
                        RecordNumber = recordNum;
                        matched = true;
                        break;
                }
        }

        return( matched );
} // end of CompareSavedRecords


/*********************************************************************************
 *
 * Function:    CalculatePeriodVitalInfo
 *
 * Description: This routine calculates statistics on the database.
 *
 * Parameters:  None
 *
 * Returns:     Nothing
 *
 * History:     09/20/00 - Added code to support database for pPatrol project.
 *
 ********************************************************************************/
static Boolean CalculatePeriodVitalInfo(void)
{
        DateType            lastDate;
        DateType            firstDate;
        Boolean             lastPeriod;
        ULong               numberDays;
        Word                numRecords;
        Word                recordNum;
        short               totalDays;
        short               goodPeriods;
        int                 theCounter;


        totalDays = 0;
        theCounter = 0;
        goodPeriods = 0;
        numberDays = 0L;
        lastPeriod = false;

        numRecords = DmNumRecords(pPatrolDB);       // Get number of database records

        for( recordNum = 0; recordNum < numRecords; recordNum++ )
        {
                FetchCurrentRecord(recordNum);

                if( !lastPeriod )
                {
                        if( TheFirst )
                        {
                                lastPeriod = true;
                                FirstDate = TheDate;
                        }
                }

                if( theCounter == 1  && TheFirst )
                {
                        firstDate = TheDate;
                        theCounter = 2;
                }

                if( TheLast )
                {
                        lastDate = TheDate;
                        theCounter = 1;
                }

                if( theCounter == 2 )
                { // Get the number of days between the last date and first date
                        numberDays = DateToDays(lastDate) - DateToDays(firstDate) + 1;

//                      ShowInformation("Counter = ", theCounter);

                        MonthDays[goodPeriods] = numberDays;
                        FirstDays[goodPeriods] = firstDate;

                        theCounter = 0;
                        goodPeriods++;
                }
        }

        // If number of good months is less than what the User selected, return false
        if( goodPeriods < SelectedMonths )
        {
                FrmAlert(NotEnoughDataAlert);
                return( false );
        }

        totalDays = 0;
        numberDays = 0;
        goodPeriods -= 1;

        for( theCounter = 0; theCounter < goodPeriods; theCounter++ )
        {
                totalDays += MonthDays[theCounter];

                // Get number of days between two first days of two different periods
                firstDate = FirstDays[theCounter];
                lastDate = FirstDays[theCounter + 1];
                numberDays += (DateToDays(firstDate) - DateToDays(lastDate));
        }

        NumberDays = numberDays / goodPeriods;

        AverageDays = totalDays / goodPeriods;

        GoodMonths = goodPeriods;     // Number of months in series that have good data

        return( true );
} // end of CalculatePeriodVitalInfo


/***************************************************************************
 *
 * Function:    CheckForLastDayOfPeriod
 *
 * Description: This routine checks the database to see if there has been a long
 *              time between the First period day and the Last period day.
 *
 * Parameters:  None
 *
 * Returns:     Nothing
 *
 * History:     09/20/00 - Added code to support database for pPatrol project.
 *
 ***************************************************************************/
static void CheckForLastDayOfPeriod(void)
{
        DateType           lastDate;
        DateType           firstDate;
        ULong              numberDays;
        Word               numRecords;
        Word               recordNum;
        int                theCounter;
        theCounter = 0;
        numRecords = DmNumRecords(pPatrolDB);        // Get number of database records

        for( recordNum = 0; recordNum < numRecords; recordNum++ )
        {
                FetchCurrentRecord(recordNum);

                if( theCounter == 0  && TheFirst )
                {
                        firstDate = TheDate;
                        theCounter++;
                }

                if( TheLast )
                {
                        lastDate = TheDate;
                        theCounter++;
                }

//              ShowInformation("Counter = ", theCounter);

                if( theCounter == 2 && CompareTwoDates(lastDate, firstDate) < 0 )
                {
                        numberDays = DateToDays(Today) - DateToDays(firstDate);

                        if( numberDays >= 11 && numberDays <= 15 ) FrmAlert(MoreThan10DaysAlert);

                        if( numberDays >= 16 && numberDays <= 20 ) FrmAlert(MoreThan15DaysAlert);

                        theCounter++;
                }
        }
} // end of CheckForLastDayOfPeriod


#pragma mark ------------------
/*****************************************************************************
 *
 * Function:    DiaryDrawCell
 *
 * Description: Draw item in the Diary Form's table. This routine is called from
 *              the table object, and must match the parameters the table object
 *              passes.  The DiaryFormLoadTable routine sets the table object to
 *              call this routine. The table object calls it once for each table
 *              cell that needs drawing.
 *
 * Parameters:  table  -> Table in which to draw the record.
 *              row    -> Row of the record to change.
 *              column -> Column of the record to change.
 *              bounds -> Bounds in which to draw the record.
 *
 * Returns:     Nothing
 *
 * History:     09/20/00 - Added code to support database for pPatrol project.
 *
 *****************************************************************************/
static void DiaryDrawCell(VoidPtr table, Word row, Word column, RectanglePtr rct)
{
        CharPtr           pMesg;
        Boolean           itFits;
        Boolean           common;
        FontID            curFont;
        Word              recordNum;
        short             posX, posY;
        short             length, width;
        char              buffer[32];
        char              noteChar;
        char              theDate[dateStringLength];


        // It's a Pilot custom to not destroy the current font, but rather to save and
        // restore the current font.
        curFont = FntSetFont(stdFont);

        // Get the record number, stored as the RowID, then retrieve the record's data
        recordNum = TblGetRowID(table, row);
        FetchCurrentRecord(recordNum);          // Get record data so we can process it

        common = false;
        posX = rct->topLeft.x;
        posY = rct->topLeft.y;

        switch( column )
        {
                case DateColumn:                        // Column 0 shows the record date
                        pMesg = MemPtrNew(15);
                        DateToAscii(TheDate.month, TheDate.day, (TheDate.year + firstYear) % 100,
DisplayDate, theDate);
                        StrCopy(pMesg, theDate);
                        pMesg[StrLen(pMesg)] = nullChr;

                        // Remove year from date string
                        if( (DisplayDate == dfYMDWithSlashes) || (DisplayDate == dfYMDWithDots) ||
                                        (DisplayDate == dfYMDWithDashes) )
                        {
                                pMesg += 3;
                        }
                        else
                                pMesg[StrLen(pMesg) - 3] = nullChr;

                        common = true;
                        break;

                case FlowColumn:                        // Column 1 shows the flow text
                        pMesg = MemPtrNew(20);

                        SysStringByIndex(DailyFlowStringList, TheFlow, buffer, sizeof(buffer));

                        StrCopy(pMesg, buffer);
                        width = rct->extent.x;
                        length = StrLen(pMesg);
                        FntCharsInWidth(pMesg, &width, &length, &itFits);

                        if( !itFits )           // If necessary, truncate characters in this cell
                        {
                                pMesg[length - 1] = 0x85;
                                pMesg[length] = chrNull;
                        }

                        common = true;
                        break;

                case MoodColumn:                        // Column 2 shows the mood text
                        pMesg = MemPtrNew(20);

                        if( TheMood == OTHER )
                                StrCopy(buffer, TheOther);
                        else
                                SysStringByIndex(DailyMoodStringList, TheMood, buffer,
sizeof(buffer));

                        StrCopy(pMesg, buffer);
                        width = rct->extent.x;
                        length = StrLen(pMesg);
                        FntCharsInWidth(pMesg, &width, &length, &itFits);

                        if( !itFits )           // If necessary, truncate characters in this cell
                        {
                                pMesg[length - 1] = 0x85;
                                pMesg[length] = chrNull;
                        }
                        common = true;
                        break;

                case NotesColumn:                               // Column 3 shows any notes written
                        if( StrLen(TheNotes) != 0 )             // Draw note symbol if record has a note
                        {
                                curFont = FntSetFont(symbolFont);
                                noteChar = symbolNote;
                                WinDrawChars(&noteChar, 1, posX, posY);
                                FntSetFont (curFont);
                        }
                        break;
        }

        if( common )
        {
                WinDrawChars(pMesg, StrLen(pMesg), posX, posY);
                FntSetFont(curFont);                                            // Restore the font
                MemPtrFree(pMesg);
        }
} // end of DiaryDrawCell


/*****************************************************************************
 *
 * Function:    DiaryLoadTable
 *
 * Description: Loads database records into the DiaryForm table.
 * Description: Loads the table object with database records. But before loading
 *              the table with records, do any needed positioning of the table.
 *
 * Parameters:  recordNum -> Index of first record to display.
 *
 * Returns:     Nothing
 *
 * History:     09/20/00 - Added code to support database for pPatrol project.
 *
 *****************************************************************************/
static void DiaryLoadTable(void)
{
        FormPtr         pForm;
        TablePtr        pTable;
        VoidHand        moniker;                        // Handle for current record
        Boolean         enableDown, enableUp;
        Word            lastRecord, recordNum;
        Word            rowNumber, rowsInTable;
        Word            recordNumber;
        int             indexDown, indexUp;


        pForm = FrmGetActiveForm();                     // Get pointer to active form
        recordNum = dmMaxRecordIndex;
        pTable = GetObjectPtr(DiaryRecordsTable);
        rowsInTable = TblGetNumberOfRows(pTable);

        // Try showing a full display of records.  Starting at last record and working
        // backwards, find record displayed at top of table.  If this record is before
        // the TopVisibleRecord then the TopVisibleRecord is set too far down the list
        // of records.  Set the TopVisibleRecord to record one screen from the end.
        DmSeekRecordInCategory(pPatrolDB, &recordNum, rowsInTable - 1, dmSeekBackward, 0);

        TopVisibleRecord = recordNumber = min(TopVisibleRecord, recordNum);

        for( rowNumber = rowsInTable - 1; (short)rowNumber >= 0; rowNumber--, recordNumber++ )
        {                                       // Get each record in the current category
                moniker = DmQueryNextInCategory(pPatrolDB, &recordNumber, 0);

                // If a record was found, set TblSetItemStyle to customTableItem, which says
                // we want to be called to draw the record.  Also store the record number as
                // the RowID and then set the row usable and mark it invalid so it will draw
                // when the draw routine is called.
                if( moniker )
                {
                        TblSetItemStyle(pTable, rowNumber, DateColumn, customTableItem);
                        TblSetItemStyle(pTable, rowNumber, FlowColumn, customTableItem);
                        TblSetItemStyle(pTable, rowNumber, MoodColumn, customTableItem);
                        TblSetItemStyle(pTable, rowNumber, NotesColumn, customTableItem);

        TblSetRowID(pTable, rowNumber, recordNumber);
                        TblSetRowUsable(pTable, rowNumber, true);
                        TblMarkRowInvalid(pTable, rowNumber);
                        lastRecord = recordNumber;
                }
                else    // If there are more rows than records, mark unused rows as unusable
                        TblSetRowUsable(pTable, rowNumber, false);
        }

        TblSetCustomDrawProcedure(pTable, DateColumn, DiaryDrawCell);
        TblSetCustomDrawProcedure(pTable, FlowColumn, DiaryDrawCell);
        TblSetCustomDrawProcedure(pTable, MoodColumn, DiaryDrawCell);
        TblSetCustomDrawProcedure(pTable, NotesColumn, DiaryDrawCell);

        TblSetColumnUsable(pTable, DateColumn, true);
        TblSetColumnUsable(pTable, FlowColumn, true);
        TblSetColumnUsable(pTable, MoodColumn, true);
        TblSetColumnUsable(pTable, NotesColumn, true);

        // If first record displayed is not last record in category, enable scroll up
        recordNum = lastRecord;
        enableUp = !DmSeekRecordInCategory(pPatrolDB, &recordNum, 1, dmSeekForward, 0);

        // If last record displayed is not first record in category enable scroll down
        recordNum = TopVisibleRecord;
        enableDown = !DmSeekRecordInCategory(pPatrolDB, &recordNum, 1, dmSeekBackward, 0);

        // Now update the on-screen scroll buttons
        indexUp = FrmGetObjectIndex(pForm, DiaryScrollUpRepeating);
        indexDown = FrmGetObjectIndex(pForm, DiaryScrollDownRepeating);
        FrmUpdateScrollers(pForm, indexUp, indexDown, enableUp, enableDown);
} // end of DiaryLoadTable


/***************************************************************************
 *
 * Function:    DiaryTableScrolling
 *
 * Description: Scrolls the list of database records in the direction specified.
 *
 *              Scrolling UP stops at the first record visible.  This is because
 *              using categories and private records the first record visible is
 *              not necessarily record 0.
 *
 *              Scrolling DOWN stops when less than a full table of records can
 *              be displayed.  To enforce this, when scrolling down, we check if
 *              at the new position there are enough records visible to fill up
 *              the table.  If not, we find the last records visible by working
 *              backwards from the end.
 *
 * Parameters:  updown  -> up or down.
 *              oneLine -> true scrolls one line, false scrolls one page.
 *
 * Returns:     Nothing
 *
 * History:     09/20/00 - Added code to support database for pPatrol project.
 *
 ***************************************************************************/
static void DiaryTableScrolling(DirectionType updown, Boolean oneLine)
{
        TablePtr            pTable = GetObjectPtr(DiaryRecordsTable);
        Word                rowsInTable = TblGetNumberOfRows(pTable);
        Word                topVisibleItem;
        CurrentRecord = NothingSelected;
        topVisibleItem = TopVisibleRecord;

        if( updown == up )                                          // Scroll table UP
        {
                if( oneLine )                                         // Scroll up one line
                {
                        DmSeekRecordInCategory(pPatrolDB, &topVisibleItem, 1, dmSeekForward, 0);
                }
                else                                    // Scroll up one page (less one row)
                { // Try going forward one page
                        if( DmSeekRecordInCategory(pPatrolDB, &topVisibleItem, rowsInTable - 1,
                                        dmSeekForward, 0) )
                        {               // Try going backwards one page from the last record
                                topVisibleItem = dmMaxRecordIndex;
                                DmSeekRecordInCategory(pPatrolDB, &topVisibleItem, rowsInTable - 1,
                                        dmSeekBackward, 0);
                        }
                }
        }
        else                                                        // Scroll table DOWN
        {
                if( oneLine )                                       // Scroll down one line
                {
                        DmSeekRecordInCategory(pPatrolDB, &topVisibleItem, 1, dmSeekBackward,
                                0);
                }
                else                                    // Scroll down one page (less one row)
                {
                        if( DmSeekRecordInCategory(pPatrolDB, &topVisibleItem, rowsInTable - 1,
                                        dmSeekBackward, 0))
                        {       // Not enough records to fill one page, so start with first record
                                topVisibleItem = 0;
                                DmSeekRecordInCategory(pPatrolDB, &topVisibleItem, 0,
dmSeekForward,
                                0);
                        }
                }
        }

        if( TopVisibleRecord != topVisibleItem )        // Avoid redraw if no changes
        {  // Table is at different position so load it with new records and redraw it
                TopVisibleRecord = topVisibleItem;

                DiaryLoadTable();                       // Setup and display Diary table
                TblRedrawTable(pTable);
        }
} // end of DiaryTableScrolling


#pragma mark -------------------
/******************************************************************************
 *
 * Function:    MainFormInitialization
 *
 * Description: Initialization routine for 'Main' form.   Does those things that
 *              need doing whenever the app starts and the 'Main' form is shown.
 *
 * Parameters:  None
 *
 * Returns:     Nothing
 *
 * History:     07/28/98 - First attempt at a generic application framework.
 *
 ******************************************************************************/
static void MainFormInitialization(void)
{
        Boolean         newDate = false;
        char            buffer[longDateStrLength];
        DateToAscii(Today.month, Today.day, Today.year + firstYear, DisplayLongDate, buffer);
        WinDrawChars(buffer, StrLen(buffer), 105, 1);             // Show today's date

        DateSecondsToDate(TimGetSeconds(), &Nuday);

        ShowMeTheDate(newDate, Nuday, MainFirstSelTrigger);
        ShowMeTheDate(newDate, Nuday, MainLastSelTrigger);

        CtlSetValue(GetObjectPtr(MainFirstCheckbox), false);
        CtlSetValue(GetObjectPtr(MainLastCheckbox), false);

        ClearFieldById(MainNotesField);
        ClearFieldById(MainMoodField);
        ClearFieldById(MainFlowField);

        DailyFlow = NothingSelected;
        DailyMood = NothingSelected;
} // end of MainFormInitialization


/*****************************************************************************
 *
 * Function:    MainFormMenuHandler
 *
 * Description: This routine performs the menu command specified by the User.
 *
 * Parameters:  command -> Menu item ID tag.
 *
 * Returns:     Nothing
 *
 * History:     07/28/98 - First attempt at my generic application framework.
 *
 *****************************************************************************/
static void MainFormMenuHandler(Word command)
{
        switch( command )
        {
                case OptionsCalendar:
                        FrmPopupForm(CalendarForm);
                        break;

                case OptionsPreferences:
                        FrmPopupForm(PreferencesForm);
                        break;

                case OptionsPeriodDiary:
                        FrmPopupForm(DiaryForm);
                        break;

                case OptionsVitalInformation:
                        FrmPopupForm(VitalInfoForm);
                        break;

                case OptionsNumberofRecords:
                        GetNumberOfRecords();
                        break;

                case OptionsDisclaimer:
                        FrmPopupForm(DisclaimerForm);
                        break;

                case OptionsAboutpPatrol:
                        FrmPopupForm(AboutAppForm);
                        break;
        }
} // end of MainFormMenuHandler


/*****************************************************************************
 *
 * Function:    MainFormEventHandler
 *
 * Description: Event handler for the application's 'MainForm'. Processes events
 *              when the main form is active.
 *
 * Parameters:  event -> Pointer to an EventType structure.
 *
 * Returns:     True if event was handled and should not be passed to a higher
 *              level handler or False (0) if the event was not handled.
 *
 * History:     07/28/98 - First attempt at my generic application framework.
 *
 ***************************************************************************/
static Boolean MainFormEventHandler(EventPtr event)
{
        Boolean         handled = false;            // Assume we might not succeed
        ControlPtr      pCntrl0 = GetObjectPtr(MainFirstCheckbox);
        ControlPtr      pCntrl1 = GetObjectPtr(MainLastCheckbox);
        FieldPtr        pField;
        ListPtr         pList;
        Word            selected;


        switch( event->eType )
        {
                case ctlSelectEvent:            // Control button was pressed and released
                        if( event->data.ctlEnter.controlID == MainFirstCheckbox )
                        {
                                if( CtlGetValue(pCntrl1) ) CtlSetValue(pCntrl1, false);
                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == MainFirstSelTrigger )
                        {
                                if( CtlGetValue(pCntrl0) )
                                        Nuday = ShowMeTheDate(true, Nuday, MainFirstSelTrigger);

                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == MainLastCheckbox )
                        {
                                if( CtlGetValue(pCntrl0) ) CtlSetValue(pCntrl0, false);
                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == MainLastSelTrigger )
                        {
                                if( CtlGetValue(pCntrl1) )
                                        Nuday = ShowMeTheDate(true, Nuday, MainLastSelTrigger);

                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == MainFlowPopTrigger )
                        {
                                pList = GetObjectPtr(MainFlowList);
                                pField = GetObjectPtr(MainFlowField);

                                selected = LstPopupList(pList);
                                if( selected != NothingSelected )
                                {
                                        DailyFlow = selected;
                                        PutTextInField(pField, LstGetSelectionText(pList,
selected));
                                }

                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == MainMoodPopTrigger )
                        {
                                pList = GetObjectPtr(MainMoodList);
                                pField = GetObjectPtr(MainMoodField);

                                selected = LstPopupList(pList);
                        if( selected != NothingSelected )
                        {
                                DailyMood = selected;

                                if( selected == OTHER )
                                {
                                        ClearFieldById(MainMoodField);
                                        SetFocusOnItem(MainMoodField);
                                }
                                else
                                        PutTextInField(pField, LstGetSelectionText(pList,
selected));
                        }

                        handled = true;
                }
                else if( event->data.ctlEnter.controlID == MainClearButton )
                {                       // 'Clear' button pressed so clear flow field
                        ClearFieldById(MainFlowField);
                        handled = true;
                }
                else if( event->data.ctlEnter.controlID == MainSaveButton )
                {                       // 'Save' button pressed so save the current data
                        if( !CtlGetValue(GetObjectPtr(MainLastCheckbox)) &&
                                        !CtlGetValue(GetObjectPtr(MainFirstCheckbox)) &&
                                        FldGetTextLength(GetObjectPtr(MainFlowField)) == 0
&&
                                        FldGetTextLength(GetObjectPtr(MainMoodField)) == 0
&&
                                        FldGetTextLength(GetObjectPtr(MainNotesField)) == 0
)
                        {
                                FrmAlert(NothingToSaveAlert);
                                handled = true;
                                break;
                        }

                        if( CompareSavedRecords() )
                        {       // Get confirmation from User before replacing the existing
record
                                if( FrmAlert(DuplicateRecordAlert) == DuplicateRecordNo )
                                {
                                        handled = true;
                                        break;
                                }
                                else
                                {
                                        if( FrmAlert(ReplaceThisRecordAlert) ==
ReplaceThisRecordNo )
                                        {
                                                handled = true;
                                                break;
                                        }
                                        else
                                        {
                                                CurrentRecord = RecordNumber;
                                        }
                                }
                        }
                        else
                                MakeNewRecord();                        // Create new
database record

                        // Grab User data into a structure so we can write it as a packed
record
                        TheDate = Nuday;

                        TheFlow = DailyFlow;

                        TheMood = DailyMood;
```

```
                                TheLast = CtlGetValue(pCntrl1);

                                TheFirst = CtlGetValue(pCntrl0);

                                TheNotes = FldGetTextPtr(GetObjectPtr(MainNotesField));
                                if( TheNotes == NULL ) TheNotes = "" + nullChr;

                                if( TheMood == OTHER ) TheOther =
FldGetTextPtr(GetObjectPtr(MainMoodField));
                                if( TheOther == NULL ) TheOther = "" + nullChr;

                                SaveCurrentRecord(CurrentRecord);          // Save data from this
date

                                FrmUpdateForm(MainForm, SomethingChanged);

                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == MainDiaryButton )
                        {
                                FrmPopupForm(DiaryForm);
                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == MainVitalButton )
                        {
                                FrmPopupForm(VitalInfoForm);
                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == MainHelpButton )
                        {
                                FrmHelp(MainFormHelpString);
                                handled = true;
                        }
                        break;

                case menuEvent:                                   // Menu item was selected
                        MenuEraseStatus(0);                      // Clear menu from display first
                        MainFormMenuHandler(event->data.menu.itemID);
                        handled = true;
                        break;

                case frmUpdateEvent:
                        MainFormInitialization();
                        handled = true;
                        break;

                case frmOpenEvent:                      // Opening the form - initialize it
                        FrmDrawForm(FrmGetActiveForm());                          // Draw the form
                        DrawInfoButton(MainHelpButton);

                        if( Prefs.lastMissing )
                                CheckForLastDayOfPeriod();

                        MainFormInitialization();
                        handled = true;
                        break;
        }

        return( handled );
} // end of MainFormEventHandler


/*********************************************************************************
 *
 * Function:    AboutAppEventHandler
 *
 * Description: Displays the application 'About' form.
 *
 * Parameters:  event -> Pointer to an EventType structure.
 *
 * Returns:     True if event was handled and should not be passed to a higher
 *             level handler or false (0) if the event was not handled.
 *
 * History:     09/18/00 - Initial creation of function for pPatrol project.
 *
 ***************************************************************************/
static Boolean AboutAppEventHandler(EventPtr event)
{
        Boolean         handled = false;        // Assume we might not succeed
        CharPtr         pName, pVersion;
        Handle          moniker;
        FontID          curFont;
        short           xPos;


        // It's a Pilot custom to not destroy the current font, but rather to save and
        // restore the current font.
        curFont = FntSetFont(stdFont);

        switch( event->eType )
        {
                case ctlSelectEvent:            // Control button was pressed and released
                        if( event->data.ctlEnter.controlID == AboutAppOKButton )
                        {                       // 'OK' button pressed so apply changes and return
                                FrmReturnToForm(MainForm);                      // Return to Main
form

                                FntSetFont(curFont);                            // Restore the
font

                                handled = true;
                        }
                        break;

                case frmOpenEvent:                      // Opening the form - initialize it
                        FrmDrawForm(FrmGetActiveForm());                        // Draw the form

                        FntSetFont(largeFont);                  // Set font so that it can be seen

                        moniker = DmGetResource('tAIN', 1000);

                        if( moniker )
                        {
                                pName = MemHandleLock(moniker);
                                MemHandleUnlock(moniker);
                                DmReleaseResource(moniker);
                        }

                        xPos = (156 - FntCharsWidth(pName, StrLen(pName))) / 2;
                        WinDrawChars(pName, StrLen(pName), xPos, 14);

                        pVersion = MemPtrNew(15);
                        StrCopy(pVersion, "Version ");

                        FntSetFont(boldFont);                   // Set font so that it can be seen

                        moniker = DmGetResource('tver', 1000);

                        if( moniker )
                        {
                                pName = MemHandleLock(moniker);
                                StrCat(pVersion, pName);
                                MemHandleUnlock(moniker);
                                DmReleaseResource(moniker);
                        }

                        xPos = (156 - FntCharsWidth(pVersion, StrLen(pVersion))) / 2;
                        WinDrawChars(pVersion, StrLen(pVersion), xPos, 30);

                        MemPtrFree(pVersion);
                        handled = true;
                        break;
        }
        return( handled );
} // end of AboutAppEventHandler


/*****************************************************************************
 *
 * Function:    DisclaimerEventHandler
 *
 * Description: Displays the application 'Disclaimer' form.
 *
 * Parameters:  event -> Pointer to an EventType structure.
 *
 * Returns:     True if event was handled and should not be passed to a higher
 *              level handler or false (0) if the event was not handled.
 *
 * History:     09/18/00 - Initial creation of function for pPatrol project.
 *
 *****************************************************************************/
static Boolean DisclaimerEventHandler(EventPtr event)
{
        Boolean         handled = false;            // Assume we might not succeed


        switch( event->eType )
        {
                case ctlSelectEvent:            // Control button was pressed and released
                        if( event->data.ctlEnter.controlID == DisclaimerOKButton )
                        {                       // 'OK' button pressed so apply changes and return
                                FrmReturnToForm(MainForm);                      // Return to Main
form

                                handled = true;
                        }
                        break;

                case frmOpenEvent:                      // Opening the form - initialize it
                        FrmDrawForm(FrmGetActiveForm());                        // Draw the form
                        handled = true;
                        break;
        }

        return( handled );
} // end of DisclaimerEventHandler


/*****************************************************************************
 *
 * Function:    PreferencesEventHandler
 *
 * Description: Handles processing of events for the 'Preferences' dialog form.
 *
 * Parameters:  event -> Pointer to an EventType structure
 *
 * Returns:     True if event was handled and should not be passed to a higher
 *              level handler or false (0) if the event was not handled.
 *
 * History:     09/18/00 - Initial creation of function for pPatrol project.
 *
 *****************************************************************************/
static Boolean PreferencesEventHandler(EventPtr event)
{
        ControlPtr      pCntrl0 = GetObjectPtr(PreferencesNextPeriodCheckbox);
        ControlPtr      pCntrl1 = GetObjectPtr(PreferencesLastDayCheckbox);
        Boolean         handled = false;            // Assume we might not succeed


        switch( event->eType )
        {
                case ctlSelectEvent:            // Control button was pressed and released
                        if( event->data.ctlEnter.controlID == PreferencesOKButton )
                        {           // 'OK' button pressed so apply changes and return to Main form
                                Prefs.nextPeriod = CtlGetValue(pCntrl0);
                                Prefs.lastMissing = CtlGetValue(pCntrl1);

                                FrmReturnToForm(MainForm);
                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == PreferencesCancelButton )
                        {               // 'Cancel' button pressed so just return to Main form
                                FrmReturnToForm(MainForm);
                                handled = true;
                        }
                        break;

                case frmOpenEvent:                      // Opening the form - initialize it
                        FrmDrawForm(FrmGetActiveForm());                        // Draw the form

                        CtlSetValue(pCntrl0, Prefs.nextPeriod);
                        CtlSetValue(pCntrl1, Prefs.lastMissing);
                        handled = true;
                        break;
        }

        return( handled );
} // end of PreferencesEventHandler


/******************************************************************************
 *
 * Function:    DiaryFormEventHandler
 *
 * Description: Handles User selecting options.
 *
 * Parameters:  event -> Pointer to an EventType structure.
 *
 * Returns:     True if event was handled and should not be passed to a higher
 *              level handler or false (0) if the event was not handled.
 *
 * History:     09/18/00 - Initial creation of function for pPatrol project.
 *
 ******************************************************************************/
static Boolean DiaryFormEventHandler(EventPtr event)
{
        Boolean         handled = false;                // Assume we might not succeed
        TablePtr        pTable = GetObjectPtr(DiaryRecordsTable);
        TablePtr        tableP;
        static Word     row, col;


        switch( event->eType )
        {
                case ctlSelectEvent:            // Control button was pressed and released
                        if( event->data.ctlEnter.controlID == DiaryFinishedButton )
                        {               // 'OK' button pressed so apply changes and return
                                FrmReturnToForm(MainForm);

                                FrmUpdateForm(MainForm, SomethingChanged);
                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == DiaryEditButton )
                        {
                                if( col == 0 )
                                {
                                        if( CurrentRecord == NothingSelected )
                                                FrmAlert(NoRecordSelectedAlert);
                                        else
                                                FrmPopupForm(EditForm);

                                        TblUnhighlightSelection(pTable);        // Unhighlight
selected row
                                }

                                handled = true;
                }
                else if( event->data.ctlEnter.controlID == DiaryHelpButton )
                {
                        FrmHelp(DiaryFormHelpString);
                        handled = true;
                }
                break;

        case tblSelectEvent:
                row = event->data.tblSelect.row;
                col = event->data.tblSelect.column;
                tableP = event->data.tblSelect.pTable;

                // Get record number from RowID then get record so we can process the data
                CurrentRecord = TblGetRowID(tableP, row);
                FetchCurrentRecord(CurrentRecord);

                if( col == 1 || col == 2 )
                {
                        TblUnhighlightSelection(pTable);            // Unhighlight
selected row
                }
                else if( col == 3 )
                {
                        if( StrLen(TheNotes) != 0 ) FrmPopupForm(NotesForm);

                        TblUnhighlightSelection(pTable);            // Unhighlight
selected row
                }

                handled = true;
                break;

        case ctlRepeatEvent:   // On-screen scroll button pressed so scroll one line
                if( event->data.ctlRepeat.controlID == DiaryScrollDownRepeating )
                {
                        DiaryTableScrolling(down, true);
                }
                else if( event->data.ctlRepeat.controlID == DiaryScrollUpRepeating )
                {
                        DiaryTableScrolling(up, true);
                }
                break;          // Repeating controls don't repeat if 'handled' set true

        case keyDownEvent:
                if( event->data.keyDown.chr == pageUpChr )       // Hard-key scroll button
                {
                        DiaryTableScrolling(up, false);
                        handled = true;
                }
                else if( event->data.keyDown.chr == pageDownChr )// Hard-key scroll button
                {
                        DiaryTableScrolling(down, false);
                        handled = true;
                }
                break;

        case frmUpdateEvent:
                DmQuickSort(pPatrolDB, (DmComparF *)CompareDateFunc, 0);

                TblEraseTable(pTable);
                DiaryLoadTable();                       // Setup and display Diary table
                TblDrawTable(pTable);

                handled = true;
                break;

        case frmOpenEvent:                      // Opening the form - initialize it
                FrmDrawForm(FrmGetActiveForm());                        // Draw the form
                DrawInfoButton(DiaryHelpButton);
```

```
                    DiaryLoadTable();                            // Setup and display Diary table
                    TblDrawTable(pTable);

                    WinDrawLine(0, 140, 159, 140);               // Draw a form separator
line

                    handled = true;
                    break;
        }

        return( handled );
} // end of DiaryFormEventHandler


/*******************************************************************************
 *
 * Function:    UserNotesEventHandler
 *
 * Description: Handles User selecting options.
 *
 * Parameters:  event -> Pointer to an EventType structure.
 *
 * Returns:     True if event was handled and should not be passed to a higher
 *              level handler or false (0) if the event was not handled.
 *
 * History:     09/18/00 - Initial creation of function for pPatrol project.
 *
 *******************************************************************************/
static Boolean UserNotesEventHandler(EventPtr event)
{
        Boolean         handled = false;            // Assume we might not succeed
        FieldPtr        pField;


        switch( event->eType )
        {
                case ctlSelectEvent:            // Control button was pressed and released
                        if( event->data.ctlEnter.controlID == NotesOKButton )
                        {                       // 'OK' button pressed so apply changes and return
                                CurrentRecord = NothingSelected;

                                FrmReturnToForm(DiaryForm);
                                handled = true;
                        }
                        break;

                case frmOpenEvent:                      // Opening the form - initialize it
                        FrmDrawForm(FrmGetActiveForm());                      // Draw the form

                        pField = GetObjectPtr(NotesNotesField);
                        PutTextInField(pField, TheNotes);
                        handled = true;
                        break;
        }

        return( handled );
} // end of UserNotesEventHandler


/*******************************************************************************
 *
 * Function:    EditDiaryEventHandler
 *
 * Description: Handles User selecting options.
 *
 * Parameters:  event -> Pointer to an EventType structure.
 *
 * Returns:     True if event was handled and should not be passed to a higher
 *              level handler or false (0) if the event was not handled.
 *
 * History:     09/18/00 - Initial creation of function for pPatrol project.
 *
 ******************************************************************************/
static Boolean EditDiaryEventHandler(EventPtr event)
{
        Boolean         handled = false;            // Assume we might not succeed
        ControlPtr      pCntrl0 = GetObjectPtr(EditFirstCheckbox);
        ControlPtr      pCntrl1 = GetObjectPtr(EditLastCheckbox);
        FieldPtr        pField;
        ListPtr         pList;
        char            buffer[20];


        switch( event->eType )
        {
                case ctlSelectEvent:            // Control button was pressed and released
                        if( event->data.ctlEnter.controlID == EditDateSelTrigger )
                        {
                                Nuday = ShowMeTheDate(true, Nuday, EditDateSelTrigger);
                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == EditFirstCheckbox )
                        {
                                if( CtlGetValue(pCntrl1) ) CtlSetValue(pCntrl1, false);
                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == EditLastCheckbox )
                        {
                                if( CtlGetValue(pCntrl0) ) CtlSetValue(pCntrl0, false);
                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == EditFlowPopTrigger )
                        {
                                pList = GetObjectPtr(EditFlowList);
                                pField = GetObjectPtr(EditFlowField);

                                DailyFlow = LstPopupList(pList);
                                if( DailyFlow != NothingSelected )
                                {
                                        PutTextInField(pField, LstGetSelectionText(pList,
DailyFlow));
                                }

                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == EditMoodPopTrigger )
                        {
                                pList = GetObjectPtr(EditMoodList);
                                pField = GetObjectPtr(EditMoodField);

                                DailyMood = LstPopupList(pList);
                                if( DailyMood != NothingSelected )
                                {
                                        if( DailyMood == OTHER )
                                        {
                                                ClearFieldById(EditMoodField);
                                                SetFocusOnItem(EditMoodField);
                                        }
                                        else
                                                PutTextInField(pField, LstGetSelectionText(pList,
DailyMood));
                                }

                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == EditSaveButton )
                        {                       // 'Save' button pressed so save the current data
                                TheDate = Nuday;

                                TheFlow = DailyFlow;

                                TheMood = DailyMood;
```

```
                              TheLast = CtlGetValue(pCntrl1);

                              TheFirst = CtlGetValue(pCntrl0);

                              TheNotes = FldGetTextPtr(GetObjectPtr(EditNotesField));
                              if( TheNotes == NULL ) TheNotes = "" + nullChr;

                              if( TheMood == OTHER ) TheOther =
FldGetTextPtr(GetObjectPtr(EditMoodField));
                              if( TheOther == NULL ) TheOther = "" + nullChr;

                              SaveCurrentRecord(CurrentRecord);          // Save data from this
date

                              FrmReturnToForm(DiaryForm);
                              CurrentRecord = NothingSelected;
                              FrmUpdateForm(DiaryForm, SomethingChanged);
                              handled = true;
                      }
                      else if( event->data.ctlEnter.controlID == EditCancelButton )
                      {               // 'Cancel' button pressed so just return to Main form
                              CurrentRecord = NothingSelected;
                              FrmReturnToForm(DiaryForm);
                              handled = true;
                      }
                      else if( event->data.ctlEnter.controlID == EditDeleteButton )
                      {                       // 'Delete' button pressed so do the dirty work
                              // Display 'DeleteRecordAlert' to get confirmation before doing a
delete
                              if( FrmAlert(DeleteRecordAlert) == DeleteRecordOK )
                              {
                                      DmRemoveRecord(pPatrolDB, CurrentRecord);
                              }

                              FrmReturnToForm(DiaryForm);
                              CurrentRecord = NothingSelected;
                              FrmUpdateForm(DiaryForm, SomethingChanged);
                              handled = true;
                      }
                      break;

              case frmOpenEvent:                     // Opening the form - initialize it
                      FrmDrawForm(FrmGetActiveForm());                        // Draw the form

                      Nuday = TheDate;               // Update current date with record date
                      ShowMeTheDate(false, Nuday, EditDateSelTrigger);

                      CtlSetValue(GetObjectPtr(EditFirstCheckbox), TheFirst);
                      CtlSetValue(GetObjectPtr(EditLastCheckbox), TheLast);

                      SysStringByIndex(DailyFlowStringList, TheFlow, buffer, sizeof(buffer));
                      PutTextInField(GetObjectPtr(EditFlowField), buffer);

                      if( TheMood == OTHER )
                              StrCopy(buffer, TheOther);
                      else
                              SysStringByIndex(DailyMoodStringList, TheMood, buffer,
sizeof(buffer));

                      PutTextInField(GetObjectPtr(EditMoodField), buffer);

                      PutTextInField(GetObjectPtr(EditNotesField), TheNotes);

                      DailyFlow = TheFlow;
                      DailyMood = TheMood;
                      handled = true;
                      break;
      }

      return( handled );
```

```
} // end of EditDiaryEventHandler


/*****************************************************************************
 *
 * Function:     PenDownCheckWhere
 *
 * Description: Handles any action necessary if, and when, the User taps within
 *              a specific area (x, y) in the display area.
 *
 * Parameters:  penX, penY -> Position origin relative to current window.
 *              marking              -- if true, pen down on game pieces is interpreted as
marking
 *              inBoundsP     -- if pen landed in game board bounds, *inBoundsP.
 *                            will be set to true, otherwise to false.
 *
 * Returns:     true if handled; false if not.
 *
 * History:     09/18/00 - Initial creation of function for pPatrol project.
 *
 ****************************************************************************/
static Boolean PenDownCheckWhere(int penX, int penY, enum events eType)
{
        RectangleType     rect;
        DateType          today;
        Boolean           handled = false;         // Assume we might not succeed
        FieldPtr          pField;
        char              buffer[longDateStrLength];


        if( FldGetTextLength(GetObjectPtr(VitalInfoMonths1Field)) != 0 )
        {
                pField = GetObjectPtr(VitalInfoDays1Field);
                FldGetBounds(pField, &rect);

                if( RctPtInRectangle(penX, penY, &rect) )
                {
                        if( CalculatePeriodVitalInfo())
                        {
                                SndPlaySystemSound(sndWarning);

                                today = FirstDate;
                                DateAdjust(&today, NumberDays);

                                pField = GetObjectPtr(VitalInfoDays1Field);
                                DateToAscii(today.month, today.day, today.year + firstYear,
DisplayDate, buffer);
                                StrCat(buffer + StrLen(buffer), ".");
                                PutTextInField(pField, buffer);
                        }

                        handled = true;
                }
        }

        if( FldGetTextLength(GetObjectPtr(VitalInfoMonths2Field)) != 0 )
        {
                pField = GetObjectPtr(VitalInfoDays2Field);
                FldGetBounds(pField, &rect);

                if( RctPtInRectangle(penX, penY, &rect) )
                {
                        if( CalculatePeriodVitalInfo())
                        {
                                SndPlaySystemSound(sndWarning);

                                StrIToA(buffer, NumberDays);
                                PutTextInField(pField, buffer);
                        }

                        handled = true;
```

```
            }
        }

        if( FldGetTextLength(GetObjectPtr(VitalInfoMonths3Field)) != 0 )
        {
                pField = GetObjectPtr(VitalInfoDays3Field);
                FldGetBounds(pField, &rect);

                if( RctPtInRectangle(penX, penY, &rect) )
                {
                        if( CalculatePeriodVitalInfo())
                        {
                                SndPlaySystemSound(sndWarning);

                                StrIToA(buffer, AverageDays);
                                StrCat(buffer + StrLen(buffer), " days.");
                                PutTextInField(pField, buffer);
                        }

                        handled = true;
                }
        }

        return( handled );
} // end of PenDownCheckWhere


/*****************************************************************************
 *
 * Function:    VitalInfoEventHandler
 *
 * Description: Handles User selecting options.
 *
 * Parameters:  event -> Pointer to an EventType structure.
 *
 * Returns:     True if event was handled and should not be passed to a higher
 *              level handler or false (0) if the event was not handled.
 *
 * History:     09/10/00 - Initial creation of function for pPatrol project.
 *
 *****************************************************************************/
static Boolean VitalInfoEventHandler(EventPtr event)
{
        DateType            today;
        Boolean             handled = false;            // Assume we might not succeed
        FieldPtr            pField;
        ListPtr             pList;
        Word                selectedMonths;
        char                buffer[longDateStrLength];


        switch( event->eType )
        {
                case ctlSelectEvent:              // Control button was pressed and released
                        if( event->data.ctlEnter.controlID == VitalInfoFinishedButton )
                        {                         // 'OK' button pressed so apply changes and return
                                FrmReturnToForm(MainForm);
                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == VitalInfoShowButton )
                        {
                                if( FldGetTextLength(GetObjectPtr(VitalInfoMonths1Field)) == 0 ||

        FldGetTextLength(GetObjectPtr(VitalInfoMonths2Field)) == 0 ||

        FldGetTextLength(GetObjectPtr(VitalInfoMonths3Field)) == 0 )
                                {
                                        FrmAlert(NoMonthsSelectedAlert);
                                        handled = true;
                                        break;
                                }
                                if( CalculatePeriodVitalInfo())
                                {
                                        today = FirstDate;
                                        DateAdjust(&today, NumberDays);

                                        pField = GetObjectPtr(VitalInfoDays1Field);
                                        DateToAscii(today.month, today.day, today.year + firstYear,
DisplayDate, buffer);
                                        StrCat(buffer + StrLen(buffer), ".");
                                        PutTextInField(pField, buffer);

                                        pField = GetObjectPtr(VitalInfoDays2Field);
                                        StrIToA(buffer, NumberDays);
                                        PutTextInField(pField, buffer);

                                        pField = GetObjectPtr(VitalInfoDays3Field);
                                        StrIToA(buffer, AverageDays);
                                        StrCat(buffer + StrLen(buffer), " days.");
                                        PutTextInField(pField, buffer);
                                }
                                else
                                { // Emulate a User tapping the 'Clear' key after having made a
mistake
                                        CtlHitControl(GetObjectPtr(VitalInfoClearButton));
                                }

                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == VitalInfoMonths0PopTrigger )
                        {
                                pList = GetObjectPtr(VitalInfoMonths0List);

                                selectedMonths = LstPopupList(pList);
                                if( selectedMonths != NothingSelected )
                                {
                                        StrCopy(buffer, LstGetSelectionText(pList,
selectedMonths));
                                        PutTextInField(GetObjectPtr(VitalInfoMonths0Field),
buffer);
                                        PutTextInField(GetObjectPtr(VitalInfoMonths1Field),
buffer);
                                        PutTextInField(GetObjectPtr(VitalInfoMonths2Field),
buffer);
                                        PutTextInField(GetObjectPtr(VitalInfoMonths3Field),
buffer);
                                        SelectedMonths = StrAToI(buffer);
                                }

                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == VitalInfoMonths1PopTrigger )
                        {
                                pList = GetObjectPtr(VitalInfoMonths1List);
                                pField = GetObjectPtr(VitalInfoMonths1Field);

                                selectedMonths = LstPopupList(pList);
                                if( selectedMonths != NothingSelected )
                                {
                                        PutTextInField(pField, LstGetSelectionText(pList,
selectedMonths));
                                        SelectedMonths = StrAToI(LstGetSelectionText(pList,
selectedMonths));
                                }

                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == VitalInfoMonths2PopTrigger )
                        {
                                pList = GetObjectPtr(VitalInfoMonths2List);
                                pField = GetObjectPtr(VitalInfoMonths2Field);
                                selectedMonths = LstPopupList(pList);
                                if( selectedMonths != NothingSelected )
                                {
                                        PutTextInField(pField, LstGetSelectionText(pList,
selectedMonths));
                                        SelectedMonths = StrAToI(LstGetSelectionText(pList,
selectedMonths));
                                }

                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == VitalInfoMonths3PopTrigger )
                        {
                                pList = GetObjectPtr(VitalInfoMonths3List);
                                pField = GetObjectPtr(VitalInfoMonths3Field);

                                selectedMonths = LstPopupList(pList);
                                if( selectedMonths != NothingSelected )
                                {
                                        PutTextInField(pField, LstGetSelectionText(pList,
selectedMonths));
                                        SelectedMonths = StrAToI(LstGetSelectionText(pList,
selectedMonths));
                                }

                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == VitalInfoClearButton )
                        {
                                ClearFieldById(VitalInfoDays1Field);
                                ClearFieldById(VitalInfoDays2Field);
                                ClearFieldById(VitalInfoDays3Field);
                                ClearFieldById(VitalInfoMonths0Field);
                                ClearFieldById(VitalInfoMonths1Field);
                                ClearFieldById(VitalInfoMonths2Field);
                                ClearFieldById(VitalInfoMonths3Field);

                                handled = true;
                        }
                        else if( event->data.ctlEnter.controlID == VitalInfoHelpButton )
                        {
                                FrmHelp(VitalInfoFormHelpString);
                                handled = true;
                        }
                        break;

                case penDownEvent:
                        handled = PenDownCheckWhere(event->screenX, event->screenY, event->eType);
                        break;

                case frmOpenEvent:                      // Opening the form - initialize it
                        FrmDrawForm(FrmGetActiveForm());                        // Draw the form
                        DrawInfoButton(VitalInfoHelpButton);
                        handled = true;
                        break;
        }

        return( handled );
} // end of VitalInfoEventHandler


#pragma mark ------------------
/*********************************************************************************
 *
 * Function:    ProtectOurApplication
 *
 * Description: Sets the bit in the database header that tells the launcher this
 *              application should not be beamable.
 *
 *              Note that this function assumes we're the active UI app.
 *             (See note in the code for what to do if you're not.)
 *
 *             Once this routine has been run, the launcher will not allow this
 *             app to be beamed. You can call this routine as many times as you
 *             want; calling it when the app is launched is convenient and will
 *             not slow down the rest of the Operating System by wasting time
 *             during the other launch codes.
 *
 *             Setting this bit at compile-time would be best, but none of the
 *             current tools allow this yet.  When they do, you can get rid of
 *             this routine.
 *
 ***************************************************************************/
static void ProtectOurApplication(void)
{
// This is a temporary definition, just in case them old headers are being used
#ifndef dmHdrAttrCopyPrevention
#define dmHdrAttrCopyPrevention  0x0040
#endif

        UInt            cardNo;
        LocalID         dbID;
        UInt            theAttributes;

        // Find our database - only works if you're the running UI application.
        // If you need to do this when you're not the running app, then call
        // DmFindDatabase() with your app's database name instead.
        SysCurAppDatabase(&cardNo, &dbID);

        if( dbID )
        {
                // Get the current attributes, turn on protection, and save them.
                DmDatabaseInfo(cardNo, dbID, 0, &theAttributes, 0,0,0,0,0,0,0,0,0);
                theAttributes = theAttributes | dmHdrAttrCopyPrevention;
                DmSetDatabaseInfo(cardNo, dbID, 0, &theAttributes, 0,0,0,0,0,0,0,0,0);
        }
} // end of ProtectOurApplication


/***************************************************************************
 *
 * Function:    CompatibleOSCheck
 *
 * Description: Check that the ROM version meets your minimum requirement.  Warn
 *              if the app was switched to by the system. This function requires
 *              a 'RomIncompatibleAlert' form resource.
 *
 * Parameters:  requiredVersion -> Minimum rom version required.
 *              (see sysFtrNumROMVersion in SystemMgr.h for format)
 *              launchFlags     -> Flags indicating how application was launched
 *              A warning is displayed only if these flags indicate that the app
 *              is launched normally.
 *
 * Returns:     Zero if OS rom is compatible else an error code.
 *
 * History:     08/19/98 - Added Operating System ROM compatibility checking.
 *
 ***************************************************************************/
static Err CompatibleOSCheck(DWord requiredVersion, Word launchFlags)
{
        DWord           romVersion;


        // See if running on minimum required version of the ROM or later.  The system
        // records the version number in a feature. A 'feature' is a specific piece of
        // information which can be looked up by a creator and feature number.
        FtrGet(sysFtrCreator, sysFtrNumROMVersion, &romVersion);
        if( romVersion < requiredVersion )
        {
                // If the User launched the app from the launcher explain why the app should
                // not be allowed to run.  If the app was contacted for something else, like
```

```
		// it was asked to find a string by the system find function, then let's not
		// bother the User with any warning dialog.  These flags tell us how the app
		// was launched to decide if a warning should be displayed.
		if( (launchFlags & (sysAppLaunchFlagNewGlobals | sysAppLaunchFlagUIApp)) ==
			(sysAppLaunchFlagNewGlobals | sysAppLaunchFlagUIApp) )
		{
			FrmAlert(RomIncompatibleAlert);

			// Pilot 1.0 will continuously relaunch this app unless we switch to
			// another safe one.  The sysFileCDefaultApp is considered "safe".
			if( romVersion < 0x02000000 )
			{
				AppLaunchWithCommand(sysFileCDefaultApp,
sysAppLaunchCmdNormalLaunch, NULL);
			}
		}

		return( sysErrRomIncompatible );
	}

	return( 0 );
} // end of CompatibleOSCheck


/***********************************************************************************
 *
 * Function:	AppHandleHotSync
 *
 * Description: Clear the backup bit after a User has done a HotSync.  This will
 *		make sure the app is backed up each and every time a User does a
 *		HotSync operation.
 *
 * Parameters:	None
 *
 * Returns:	Nothing
 *
 * History:	07/28/98 - First attempt at my generic application framework.
 *
 ***********************************************************************************/
static void AppHandleHotSync(void)
{
	DmSearchStateType dbState;
	LocalID		dbID;
	UInt		attributes, cardNo;


	// Find application database if one exists
	if( DmGetNextDatabaseByTypeCreator(true, &dbState, AppDbType, AppCreator, false, &cardNo,
&dbID) == 0 )
	{
		DmDatabaseInfo(cardNo, dbID, NULL, &attributes,
NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);

		attributes &= !dmHdrAttrBackup;

		DmSetDatabaseInfo(cardNo, dbID, NULL, &attributes,
NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);
	}
} // end of AppHandleHotSync


/***********************************************************************************
 *
 * Function:	StartApplication
 *
 * Description: Initializes anything the program needs at startup, then switches
 *		to the application's main form.  Opens database/load first form.
 *
 * Parameters:	None
 *
 * Returns:	Returns error code if there's an error or false (0) if no error.
 *
 * History:     07/28/98 - First attempt at my generic application framework.
 *
 ***************************************************************************/
static Boolean StartApplication(void)
{
        SystemPreferencesType sysPrefs;                  // User's Palm preferences
        Word            prefsSize;


        // Get current date formats from system preferences, then get today's date and
        // use this date for all records during this session unless modified by User.
        PrefGetPreferences(&sysPrefs);
        DisplayDate = sysPrefs.dateFormat;
        DisplayLongDate = sysPrefs.longDateFormat;
        StartingDayOfWeek = sysPrefs.weekStartDay;

        DateSecondsToDate(TimGetSeconds(), &Nuday);
        Today = Nuday;

        CreateApplicationDatabase();              // This should be self-explanatory

        TopVisibleRecord = 0;
        CurrentRecord = NothingSelected;
        SelectedMonths = NothingSelected;

        prefsSize = sizeof(Prefs);
        // Check if preferences have already been set and saved
        if( PrefGetAppPreferences(AppCreator, AppPrefVer, &Prefs, &prefsSize, true)
                    == noPreferenceFound )
        {
                Prefs.nextPeriod = false;
                Prefs.lastMissing = false;
                Prefs.installedDate = Nuday;
        }

        FrmGotoForm(MainForm);

        return( false );
} // end of StartApplication


/***************************************************************************
 *
 * Function:     StopApplication
 *
 * Description: If needed, save current application state and close all forms as
 *              well as any open databases.
 *
 * Parameters:  None
 *
 * Returns:     Nothing
 *
 * History:     07/28/98 - First attempt at my generic application framework.
 *
 ***************************************************************************/
static void StopApplication(void)
{
        // Close all open forms to allow their frmCloseEvent handlers to execute.  The
        // appStopEvent doesn't send frmCloseEvents, but FrmCloseAllForms does.
        FrmCloseAllForms();

        // Write saved preferences/saved-state information.  This is the data backed
        // up during a HotSync session.
        PrefSetAppPreferences(AppCreator, AppPrefVer, AppVersion, &Prefs, sizeof(Prefs), true);
} // end of StopApplication


/***************************************************************************
 *
 * Function:     EventLoop
 *
 * Description: Gets next event and hands it off to each event handler in line
 *              till one of them does something with it.  It will stay in this
 *              loop until a stop event occurs.
 *
 * Parameters:  None
 *
 * Returns:     Nothing
 *
 * History:     07/28/98 - First attempt at my generic application framework.
 *              11/02/98 - Incorporated the ApplicationEventHandler code.
 *
 ****************************************************************************/
static void EventLoop(void)
{
        EventType           event;
        FormPtr             pForm;
        Word                error, formId;


        // This is where ye old application spends most of its time just getting them
        // there events an' dispatching 'em.
        do
        {
                EvtGetEvent(&event, evtWaitForever);            // Get next available event

                if( SysHandleEvent(&event) ) continue;

                if( MenuHandleEvent(0, &event, &error) ) continue;

                if( event.eType == frmLoadEvent )
                {               // Load form resource specified in event, then activate form
                        formId = event.data.frmLoad.formID;                 // Get form ID number
                        pForm = FrmInitForm(formId);             // Load it, getting form's pointer
                        FrmSetActiveForm(pForm);            // Now OS will send events to this form

                        // Set event handler for the form.  The handler of the currently active
                        // form is called by FrmHandleEvent each time it receives an event.
                        switch( formId )
                        {
                                case MainForm:
                                        FrmSetEventHandler(pForm, MainFormEventHandler);
                                        break;

                                case PreferencesForm:
                                        FrmSetEventHandler(pForm, PreferencesEventHandler);
                                        break;

                                case AboutAppForm:
                                        FrmSetEventHandler(pForm, AboutAppEventHandler);
                                        break;

                                case DisclaimerForm:
                                        FrmSetEventHandler(pForm, DisclaimerEventHandler);
                                        break;

                                case DiaryForm:
                                        FrmSetEventHandler(pForm, DiaryFormEventHandler);
                                        break;

                                case NotesForm:
                                        FrmSetEventHandler(pForm, UserNotesEventHandler);
                                        break;

                                case EditForm:
                                        FrmSetEventHandler(pForm, EditDiaryEventHandler);
                                        break;

                                case VitalInfoForm:
                                        FrmSetEventHandler(pForm, VitalInfoEventHandler);
                                        break;
                                case CalendarForm:
                                        FrmSetEventHandler(pForm, CalendarEventHandler);
                                        break;
                        }
                }

                FrmDispatchEvent(&event);                               // Events for current form
        }
        while( event.eType != appStopEvent );
        // User chose another application, so return to PilotMain for tidyup and exit.
} // end of EventLoop


/*******************************************************************************
 *
 * Function:    PilotMain
 *
 * Description: Called by the Palm Operating System to start the application.
 *
 * Parameters:  cmd         -> Launch code; how/why application was started.
 *              cmdPBP      -> Parameter block for the command.
 *              launchFlags -> Additional flags.
 *
 * Returns:     0 for success or an applicable error code should an error occur.
 *
 * History:     07/28/98 - First attempt at my generic application framework.
 *              08/19/98 - Added Operating System version compatibility check.
 *
 ******************************************************************************/
DWord PilotMain(Word cmd, Ptr cmdPBP, Word launchFlags)
{
        Word            error;


        // This application makes use of PalmOS 2.0 features.  It will crash if run on
        // an earlier version of PalmOS.  Detect, and warn if this happens, then exit.
        error = CompatibleOSCheck(MinOSVersion, launchFlags);
        if( error ) return( error );

        if( cmd == sysAppLaunchCmdNormalLaunch )            // Check for normal launch
        {
                ProtectOurApplication();                     // Don't allow us to be beamed

                error = StartApplication();                     // Setup and initialization
                if( error ) return( error );

                EventLoop();                                    // Do the event loop boogie
                StopApplication();                          // Do any clean-up before exiting
        }

        return( error );
} // end of PilotMain
```